Course: FYS4220
Real time and embedded systems

Group number.:

Laboratory assignment 1

Combinational logic and synchronous process

August 20, 2015

DUE DATE: 15. September 2015

Completed by:

Date:

Name:
E-mail:

Name:
E-mail:

Requirements:

- **ONLY** the following files/documents are to be submitted:
  - Report: *fys4220_report_groupX_name1_name2.pdf*
  - Final design file containing all the relevant sub-assignments: *lab1.vhd*
- The report must
  - be in pdf format,
  - be well structured,
  - contain a coverpage similar to the coverpage of this document,
  - describe what you have done and any problems you have encountered during the work,
  - answer any questions which are listed in the assignment text,
  - include figures with screenshots of the various wave diagrams produced when simulating the 7-segment decoder and 4-bit counter.
- Submitted requested files/documents by e-mail to the course responsible.

Contents

1 Quartus project, combinatorial logic and functional simulation **2**
1.1 Switches and LEDs 2
1.2 7-segment decoder with functional simulation 9

2 Synchronous process **14**
2.1 4-bit counter and push button 14
2.2 Edge detector 19

Introduction

This lab assignment contains two parts. In the first part you will implement a simple design to connect the onboard toggle switches to the onboard LEDs, and a 7-segment decoder to show the value of the first 4 toggle swithces on the 7-segment display. To verify the correct behaviour of the 7-segment decoder you will simulate the design using a test bench setup which has already been prepared for you.

For the second part, you will implement a 4-bit counter that will be used as input to the 7-segment decoder. The counter will be synchronous to the onboard 50 MHz crystal, and a push butten will be used to activate the counter. A push button will be used to increment the counter. To verify the correct behaviour of the 4-bit counter you will again use an already prepared test bench setup.

1 Quartus project, combinatorial logic and functional simulation

1.1 Switches and LEDs

The purpose of this first assignment is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches on the DE1-SoC[1, 2] board as inputs to the circuit and we will connect the outputs to the light emitting diodes (LEDs). The DE1-SoC board provides 10 toggle switches, called SW9-0, which can be used as inputs to a circuit, and 10 red LEDs, called LEDR9-0, that can be used to display output values. Listing 1 shows a simple VHDL description where you will need to insert the appropriate statement in order to connect the switches to the LEDs. Since there are 10 switches and 10 LEDs it is convenient to represent them as arrays in the entity description of the VHDL code.

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;

entity lab1 is
  port (
    LEDR : out std_logic_vector(9 downto 0);   -- Red LEDs
    SW   : in  std_logic_vector(9 downto 0));  -- Switches
end entity lab1;

architecture top_level of lab1 is
begin
-- Insert statement to assign all switches to the LEDs
end architecture top_level;
```

Listing 1: Top level VHDL code for connecting the switch inputs to the LED outputs.

Perform the following steps to implement a circuit corresponding to the VHDL code in Listing 1 on the DE1-SoC board.

Create a new project:

1. Create a new directory called lab1 (at a suitable location, such as in your UiO home directory, which is backed up every day). Inside this directory create another directly called quartus.
2. Start the Quartus II program
3. Create a new Quartus II project (from File -> New Project Wizard) called lab1 and choose lab1/quartus as the working directory . The name of the top-level design will automatically take the same name as the project. Keep this name. Press Next (Figure 1).
4. For page 2 of 5 press Next without adding any files. (Design files will be added later).
5. For page 3 of 5, (Figure 2), select the correct FPGA model: Cyclone V SE (5CSEMA5F31C6) as the target chip. Press Next.
6. For page 4 of 5 you should select Modelsim-Altera under Simulation Tool name and VHDL under Simulation format. Leave the other options as illustrated in Figure 3 and press Next.
7. Page 5 of 5 now shows a summary your settings. Press Finish to create the project.

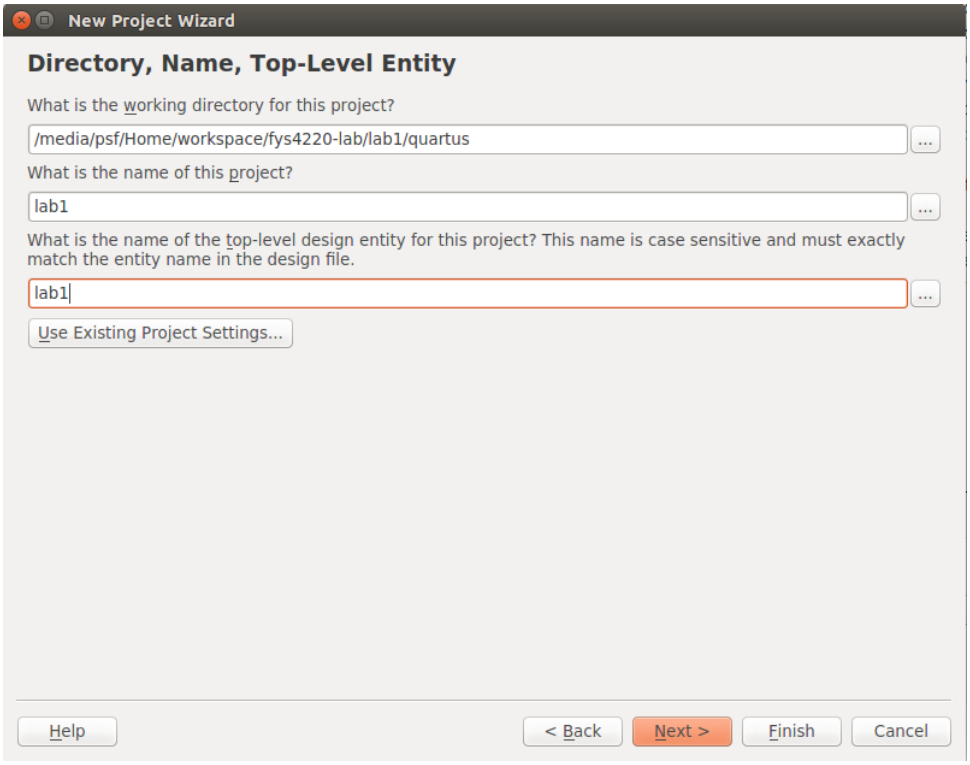
New Project Wizard
Directory, Name, Top-Level Entity
What is the working directory for this project?
/media/psf/Home/workspace/fys4220-lab/lab1/quartus
What is the name of this project?
lab1
What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
lab1
Use Existing Project Settings...
Help
< Back
Next >
Finish
Cancel

Figure 1: Provide information about project location and project name.

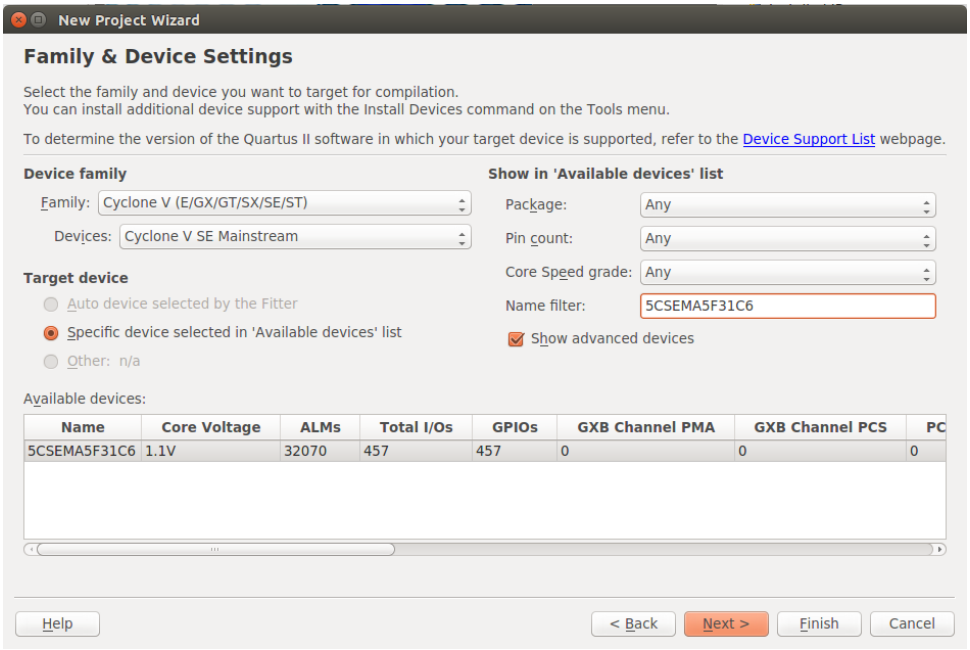

Figure 2: Choose the correct FPGA device.

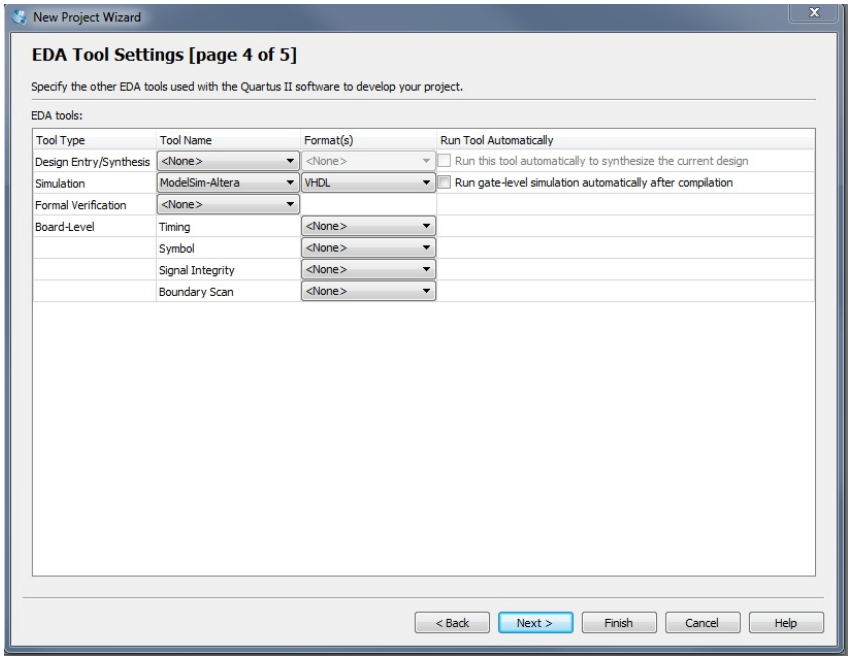

Figure 3: Select ModelSim-Altera and VHDL under Simulation and press next.

Create a new VHDL file:

1. Inside the directory lab1 create a new directory called vhd_src.
2. Open your favourite text editor (e.g. Notepad++), enter the VHDL description as shown in Listing 1 and add the appropriate statement for controlling the LEDs using the switches. Save the file in the directory lab1/vhd_src and give it the name lab1.vhd
3. Add the new VHDL file to the project as shown in Figure 4 .(Project -> Add/Remove Files in Project).

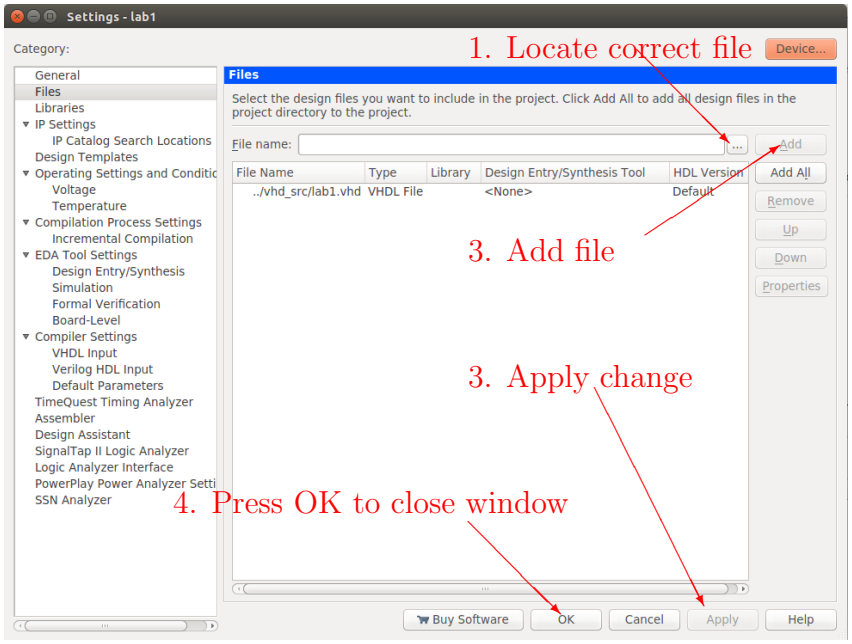

Figure 4: Instructions for adding new file to project.

Create tcl scripting file and make pin assignemnts:

The DE1-SoC board has hardwired connections between its FPGA chip and the switches and LEDs. To use SW9-0 and LEDR9-0 it is necessary to include in your Quartus II project the correct pin assignments, which are given in the DE1-SoC User Manual[2]. For example, the manual specifies that SW(0) is connected to the FPGA PIN_AB12 and LEDR(0) is connected to PIN_V16. Each pin can be assigned manually through the Quartus II pin assignment manager, however, a more elegant approach is to make the pin assignment using a tcl scripting file.

Create and import pin settings:

1. Inside the directory lab1 create a new directory called constraints.
2. Open your favourite text editor (e.g. Notepad++), enter the pinning constraints as shown in Listing 2. Save the file in the directory lab1/constraints and give it the name pinning.tcl.
3. Add the new tcl file to the project (Project -> Add/Remove Files in Project).
4. Run the tcl script to make the pin assignments (Tools -> Tcl Scripts). The tcl script should be visible under the project folder. Mark the file and click RUN. Verify that the correct assignments have been performed by opening the pin assignment editor (Assignments -> Assignment editor). You should see a similar list as shown in Figure 5.

```
#Toggle switches
set_location_assignment PIN_AB12 -to SW[0]
set_location_assignment PIN_AC12 -to SW[1]
set_location_assignment PIN_AF9  -to SW[2]
set_location_assignment PIN_AF10 -to SW[3]
set_location_assignment PIN_AD11 -to SW[4]
set_location_assignment PIN_AD12 -to SW[5]
set_location_assignment PIN_AE11 -to SW[6]
set_location_assignment PIN_AC9  -to SW[7]
set_location_assignment PIN_AD10 -to SW[8]
set_location_assignment PIN_AE12 -to SW[9]
#LED outputs
#Red LEDs
set_location_assignment PIN_V16 -to LEDR[0]
set_location_assignment PIN_W16 -to LEDR[1]
set_location_assignment PIN_V17 -to LEDR[2]
set_location_assignment PIN_V18 -to LEDR[3]
set_location_assignment PIN_W17 -to LEDR[4]
set_location_assignment PIN_W19 -to LEDR[5]
set_location_assignment PIN_Y19 -to LEDR[6]
set_location_assignment PIN_W20 -to LEDR[7]
set_location_assignment PIN_W21 -to LEDR[8]
set_location_assignment PIN_Y21 -to LEDR[9]

#To avoid that the FPGA is driving an unintended value on pins that are not in use:
set_global_assignment -name RESERVE_ALL_UNUSED_PINS "AS INPUT TRI-STATED"
```

Listing 2: Pinning information for LEDs and switches.

	Status	From	To	Assignment Name	Value	Enabled
1	Ok		SW[0]	Location	PIN_AB12	Yes
2	Ok		SW[1]	Location	PIN_AC12	Yes
3	Ok		SW[2]	Location	PIN_AF9	Yes
4	Ok		SW[3]	Location	PIN_AF10	Yes
5	Ok		SW[4]	Location	PIN_AD11	Yes
6	Ok		SW[5]	Location	PIN_AD12	Yes
7	Ok		SW[6]	Location	PIN_AE11	Yes
8	Ok		SW[7]	Location	PIN_AC9	Yes
9	Ok		SW[8]	Location	PIN_AD10	Yes
10	Ok		SW[9]	Location	PIN_AE12	Yes
11	Ok		LEDR[0]	Location	PIN_V16	Yes
12	Ok		LEDR[1]	Location	PIN_W16	Yes
13	Ok		LEDR[2]	Location	PIN_V17	Yes
14	Ok		LEDR[3]	Location	PIN_V18	Yes
15	Ok		LEDR[4]	Location	PIN_W17	Yes
16	Ok		LEDR[5]	Location	PIN_W19	Yes
17	Ok		LEDR[6]	Location	PIN_Y19	Yes
18	Ok		LEDR[7]	Location	PIN_W20	Yes
19	Ok		LEDR[8]	Location	PIN_W21	Yes
20	Ok		LEDR[9]	Location	PIN_Y21	Yes

Figure 5: Assignment editor view

Compile the project and program the FPGA:

1. Compile the project: (Processing -> Start Compilation)
2. During compilation of the project you will see some warnings related to missing constraint information.sdc file:

   - Some pins have incomplete I/O assignments
   - No clocks defined in design
   - Synopsys Design Constraints File file not found.

   These warnings can for the moment be ignored.
3. If the compilation of the project was successful, Quartus has generated and SRAM object file (.sof) either in the project directory itself or in the directory output-files under the project directory. This is the programming file that will be downloaded to the FPGA. Make sure to connect the USB cable to the USB connector on the DE1-SoC board marked with BLASTER. Connect the power cable and turn on the power by pressing the red button.
4. Open the Quartus programmer (Tools -> Programmer).
5. If the field next to the button "Hardware Setup" shows "No Hardware", make sure the USB cable is connect to both the PC and the DE1-SoC board, and that the power is turned on. Press the "Hardware Setup and choose "USB-blaser" under the "Currently selected hardware". Press Close.
6. Pressing "Auto Detect" will bring up a small window, as seen in Figure 6 where you will have to specify the correct device: 5CSEMA5. Press OK. As shown in 7 two devices are detected in the JTAG chain. One is the Hard Processor System (HPS) of the Cyclone V SoC FPGA (ARM CPU) and the other is the programmable logic part of the same FPGA.

7. Double click on the 5CSEMA5 device and select the correct programming file. Tick the box for "Programming/Configure". You should now end up with a setup as shown in 8.
8. To program the FPGA press Start.
9. Verify that your design works by changing the position of the toggle switches on the DE1-SoC board.

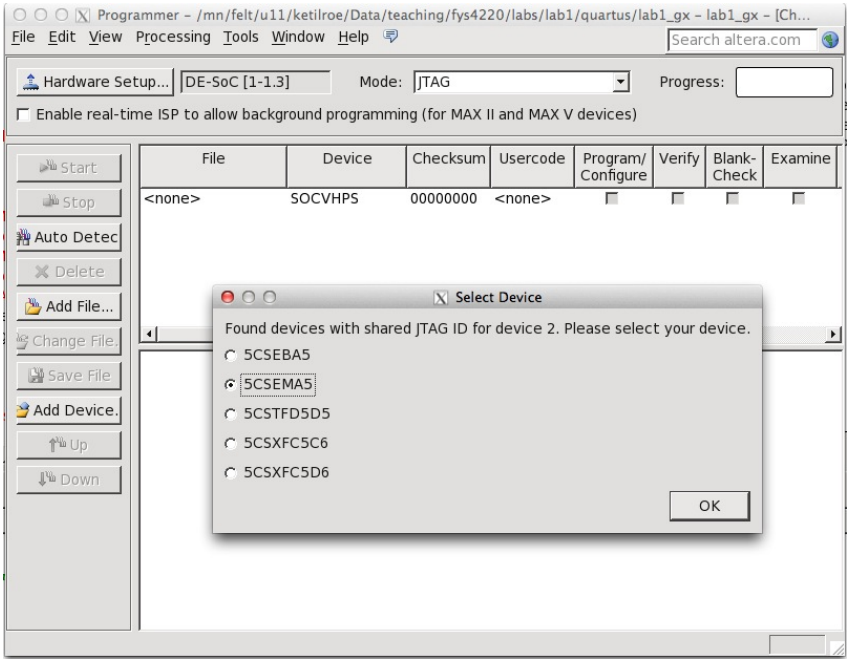

Figure 6: Press "Auto Detect" and choose the correct device: 5CSEMA5.

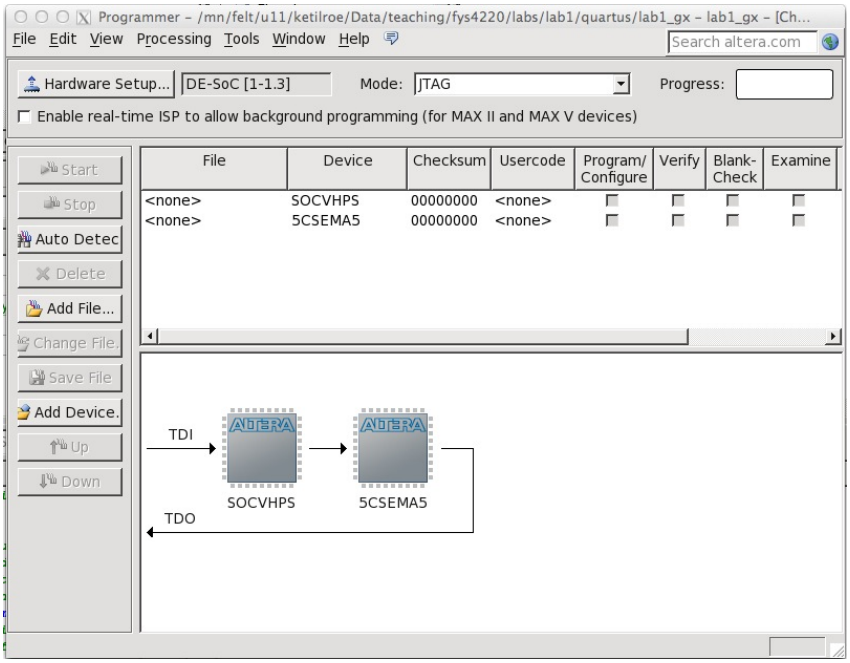

Figure 7: Two devices are detected in the JTAG chain.

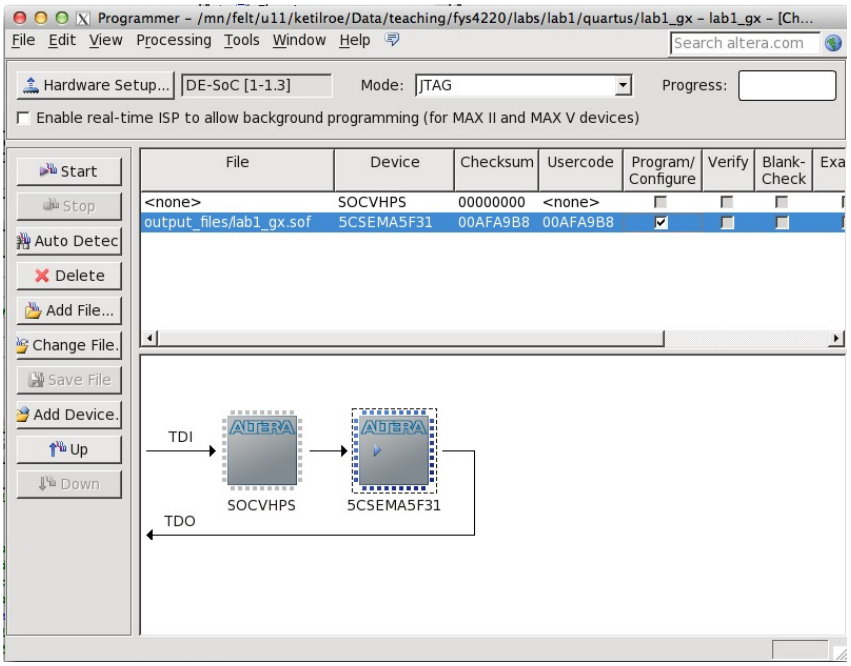

Figure 8: Final setup for programming of the FPGA. To program the FPGA press Start.

1.2 7-segment decoder with functional simulation

The objective of this part is to control one of the 7-segment displays using four switches on the DE1-SoC board. For this you need to implement a 7-segment decoder in VHDL. Before programming the board, you will verify the correct behaviour of the design using a functional simulation.

Preparing the project:

1. The 7-segment decoder (see FYS4220 lecture 2 [3]) shall be added as a statement in the lab1.vhd design file and implemented using selected signal assignment (with-select).
2. You also need to modify the entity description by adding an additional output port vector (7 bits wide) using the identifier name HEX0.
3. The four inputs of the decoder will be controlled by the toggle switches SW(3 downto 0).
4. In order to make sure that the new output ports will be connected to the correct pins, you need to add the pin assignment listed in Listing 3 to the pinning.tcl file. After saving the files run the tcl script (Tools -> Tcl Script). Again, verify the the correct pin assignments have been performed in the pinning assignment editor.

```
#Seven segment display
set_location_assignment PIN_AE26 -to HEX0[0]
set_location_assignment PIN_AE27 -to HEX0[1]
set_location_assignment PIN_AE28 -to HEX0[2]
set_location_assignment PIN_AG27 -to HEX0[3]
set_location_assignment PIN_AF28 -to HEX0[4]
set_location_assignment PIN_AG28 -to HEX0[5]
set_location_assignment PIN_AH28 -to HEX0[6]
```

Listing 3: Pin assignments for HEX display.

Simulating the design:

1. Before downloading the new design to the FPGA we want to verify that the design is working as expected. For this purpose we will be using the test bench design shown in Listing 4 . Create a new file called tb_lab1.vhd in the directory vhd_src, add the test bench design, and save the file.
2. Open Modelsim and create a new project (File-> New -> Project) as shown in Figure 9.
3. When asked to add items to the project choose "Add Existing File", browse to the vhd_src directory and add both the lab1.vhd and tb_lab1.vhd file as shown in Figures 10 and 11.
4. The files have to be compiled in the correct ordert. E.g. tb_lab1.vhd. is dependent on lab1.vhd, lab1.vhd therefore has to be compiled first. To compile right click on the file and choose (Compile -> Compile Selected).
5. To start the simulation (Simulate -> Start Simulation) and choose the design unit tb_lab1.vhd under the work library as shown in Figure 14.
6. Open the wave window if not already opened (View -> Wave).
7. Add the signals SW, HEX0, and LEDR to the wave window by marking them in the objects window, right click and choose "Add Wave" as shown in Figure 13
8. Run the simulation and verify that the design is working as expected by checking that values of HEX0 and SW. You should see a similar result as shown in Figure 15.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_lab1 is
end;

architecture testbench of tb_lab1 is
  signal LEDR : std_logic_vector(9 downto 0);
  signal SW   : std_logic_vector(9 downto 0);
  signal HEX0 : std_logic_vector(6 downto 0);

  component lab1 is
    port (
      LEDR : out std_logic_vector(9 downto 0);
```

```
      SW   : in  std_logic_vector(9 downto 0);
      HEX0 : out std_logic_vector(6 downto 0));
  end component lab1;

begin

  UUT : lab1
    port map(
      LEDR => LEDR,
      SW   => SW,
      HEX0 => HEX0
      );

  stimuli_process : process
  begin
    --set '1' as the default values for input SW.
    SW(9 downto 0) <= (others => '0');
    wait for 20 ns;
    --Run through all combinations of input values
    --for SW(3 downto 0) and validate the expected value
    --of HEX0. Chang value every 20 ns.
    SW(3 downto 0) <= X"0";
    wait for 20 ns;
    SW(3 downto 0) <= X"1";
    wait for 20 ns;
    SW(3 downto 0) <= X"2";
    wait for 20 ns;
    SW(3 downto 0) <= X"3";
    wait for 20 ns;
    SW(3 downto 0) <= X"4";
    wait for 20 ns;
    SW(3 downto 0) <= X"5";
    wait for 20 ns;
    SW(3 downto 0) <= X"6";
    wait for 20 ns;
    SW(3 downto 0) <= X"7";
    wait for 20 ns;
    SW(3 downto 0) <= X"8";
    wait for 20 ns;
    SW(3 downto 0) <= X"9";
    wait for 20 ns;
    SW(3 downto 0) <= X"A";
    wait for 20 ns;
    SW(3 downto 0) <= X"B";
    wait for 20 ns;
    SW(3 downto 0) <= X"C";
    wait for 20 ns;
    SW(3 downto 0) <= X"D";
    wait for 20 ns;
    SW(3 downto 0) <= X"E";
    wait for 20 ns;
    SW(3 downto 0) <= X"F";
    wait;
  end process stimuli_process;
end architecture testbench;
```

Listing 4: Some caption

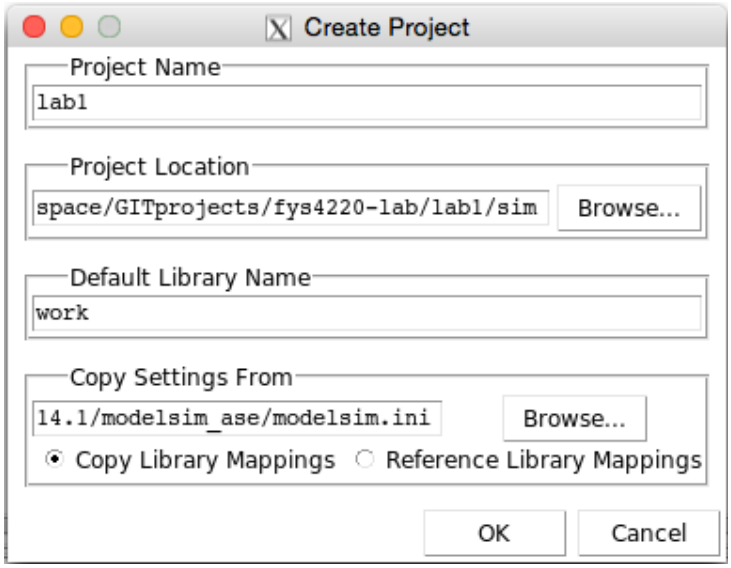

Figure 9: Creating a project in Modelsim.

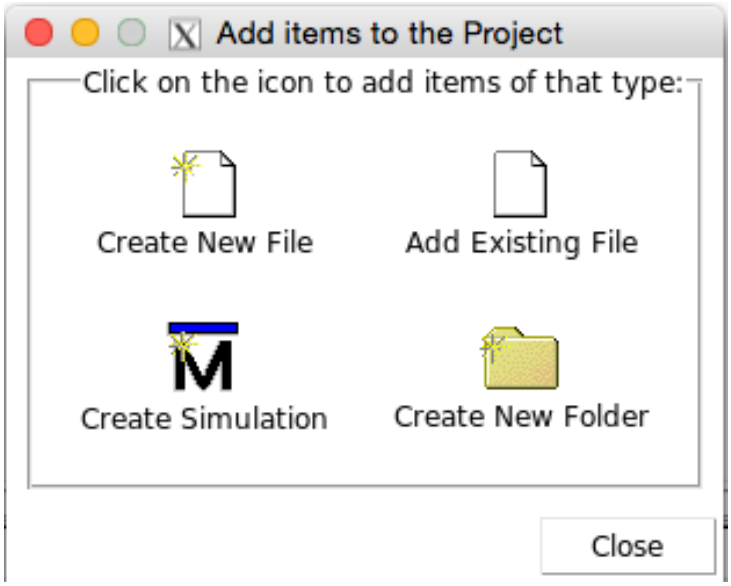

Figure 10: Choose "Add Existing File"

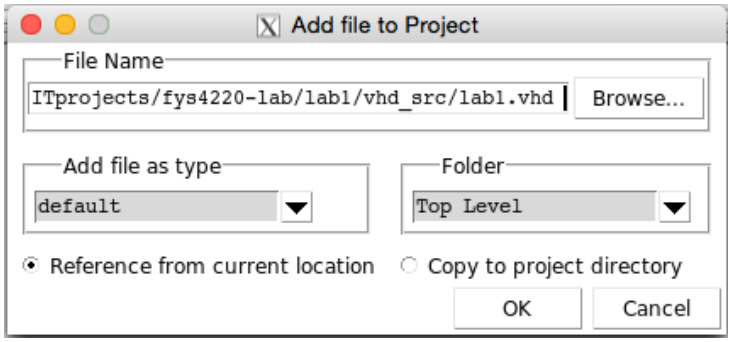

Figure 11: Add lab1.vhd and tb_lab1.vhd from the vhd_src folder.

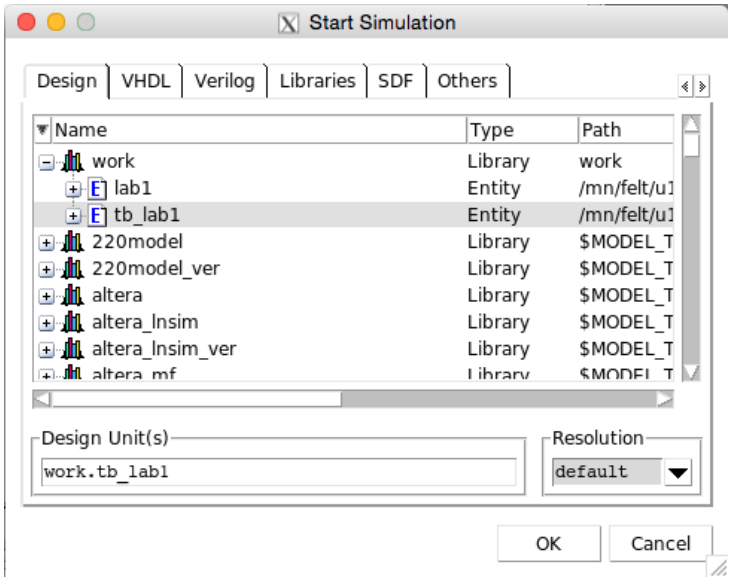

Figure 12: Choose the correct design unit to simulate.

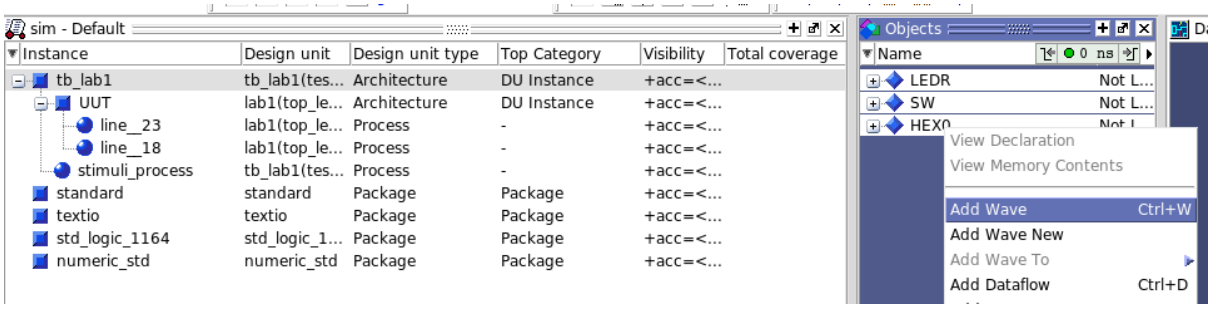

Figure 13: Add signals to the wave window.

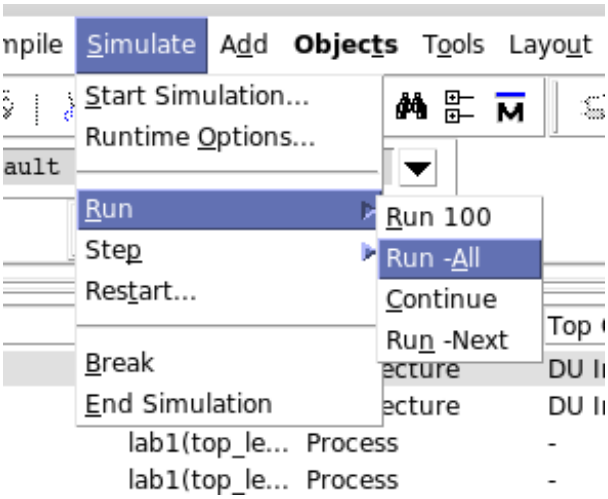

Figure 14: Choos "Run -all" under Run Simulation.

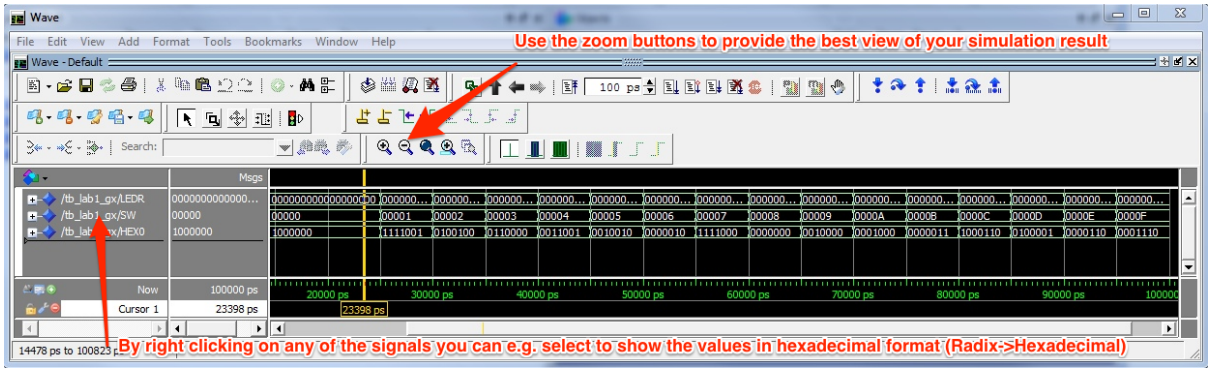
Use the zoom buttons to provide the best view of your simulation result
By right clicking on any of the signals you can e.g. select to show the values in hexadecimal format (Radix->Hexadecimal)

Figure 15: Simulation result.

Program the FPGA:

1. Return to Quartus, compile the design and program the FPGA. Verify that you can use the switches to set the correct number on the 7-segment display.

2 Synchronous process

2.1 4-bit counter and push button

In the second part of this lab. assignment you will implement a counter and use one of the push button KEY1 on the DE1-SoC card to increase the counter value in steps of one. The counter value will be shown on the 7-segment display. A second push button KEY0 will be used to reset the counter. The reset will be synchronous to the system clock.

Create the 4-bit counter with active low synchronous reset:

1. Modify the entity of lab1 and add the following ports:
   - `clk50 : in std_logic_vector`
   - `reset_n : in std_logic_vector`
   - `ext_ena_n : in std_logic`
2. Declare the internal signal:
   - `signal counter : unsigned(3 downto 0) := "0000";`
3. Write a VHDL process for the 4 bit counter. The counter value should be increased in steps of 1 (counter <= counter + 1) on the rising edge of the clk and only when the ext_ena_n = '0'.
   - Include the library numeric_std.all in order to allow for arithmetic operations and the use of the type unsigned.
4. Modify the test bench file tb_lab1.vhd in order to simulate the functional behaviour of the counter, see 5.
   - You need to update the component and port map of lab1 with the relevant new ports, add the corresponding test bench signals, generate a 50 MHz clock, and produce the relevant stimuli to test the counter.

5. Run a functional simulation and verify that the output HEX0 counts up as expected.
6. Before trying the new design on the DE1-SoC card, you need to first add the correct pinning assignments for the new ports. Add the assignments as shown in Listing 6. to the pinning.tcl file and re-run the tcl script. Verify that the new pinning assignments have been performed.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_lab1 is
end;

architecture testbench of tb_lab1 is
  signal LEDR         : std_logic_vector(9 downto 0);
  signal SW           : std_logic_vector(9 downto 0);
  signal HEX0         : std_logic_vector(6 downto 0);
  signal clk50        : std_logic;
  signal reset_n      : std_logic;
  signal ext_ena_n    : std_logic;
  signal clk_ena      : boolean;
  constant clk_period : time := 20 ns;  -- 50 MHz

  component lab1 is
    port (
      LEDR      : out std_logic_vector(9 downto 0);
      SW        : in  std_logic_vector(9 downto 0);
      HEX0      : out std_logic_vector(6 downto 0);
      clk50     : in  std_logic;
      reset_n   : in  std_logic;
      ext_ena_n : in  std_logic
      );
  end component lab1;

begin

  UUT : lab1
    port map(
      LEDR      => LEDR,
      SW        => SW,
      HEX0      => HEX0,
      clk50     => clk50,
      reset_n   => reset_n,
      ext_ena_n => ext_ena_n
      );

  -- create a 50 MHz clock
  clk50 <= not clk50 after clk_period/2 when clk_ena else '0';

  stimuli_process : process
  begin
    --2ab
    --set default values
    clk_ena   <= false;
    reset_n   <= '1';
```

```
    ext_ena_n <= '1';
    SW        <= (others => '0');

    --enable clk and wait for 3 clk periods
    clk_ena   <= true;
    wait for 3*clk_period;
    --assert reset_n for 3 clk periods
    reset_n   <= '0';
    wait for 3*clk_period;
    --deassert reset_n for 3 clk periods
    reset_n   <= '1';
    wait for 3*clk_period;
    --enable counter and wait for 20 clk_periods
    ext_ena_n <= '0';
    wait for 20*clk_period;
    --assert reset_n for 3 clk periods
    reset_n   <= '0';
    wait for 3*clk_period;
    --deassert reset_n for 10 clk periods
    reset_n   <= '1';
    wait for 3*clk_period;
    --disable clk
    clk_ena   <= false;
    --end of simulation
    wait;
  end process stimuli_process;
end architecture testbench;
```

Listing 5: Some caption

```
#50MHz clock
set_location_assignment PIN_AF14 -to clk50
#External asynchronous inputs
#Push buttons for external reset and enable
#key0
set_location_assignment PIN_AA14 -to reset_n
#key1
set_location_assignment PIN_AA15 -to ext_ena_n
```

Listing 6: Pinning information for clk, reset, and counter enable signal.

Timing analysis:

1. Compile the full project.
2. In addition to the warnings seen when compiling the first project in this lab. assignment, you will now also see the warning "Timing requirements not met". This warning indicates that the design (in most cases) now has a register-to-register path that exceeds the clock period. However, we have not yet specified the actual clock period for our design and a default clock period of 1000 MHz have therefore been set by Quartus as a default value. Verify this setting by opening the compilation report and navigate to the "Clocks" part of the TimeQuest Timing Analyzer. "Processing -> Compilation Report" (Ctrl+R).

3. Timing settings are critically important for a successful design. For this tutorial we will specify the correct clock frequency of 50 MHz by creating a basic Synopsys Design Constraints File (.sdc) that the Quartus II TimeQuest Timing Analyzer will use during design compilation. A Synopsys Design Constraints File is required by the TimeQuest Timing Analyzer to get proper timing constraints. Without it, the Compiler will not properly optimize the design. To create an SDC, perform the following steps:

   - Open the TimeQuest Timing Analyzer by choosing Tools -> TimeQuest Timing Analyzer.
   - Create the timing netlist by double clicking on "Create Timing Netlist".
   - Choose File -> New SDC file. The SDC editor opens.
   - Choose "Edit -> Insert constraint -> Create clock". This will open a new window as seen in Figure 17.
   - Set the clock name to be the same as the name of the clock input port of the design: clk50.
   - Constrain the clock to 50 MHz by setting a clock period of 20 ns.
   - The constraint needs to be identified with its correct port. Press the button next to the Target field and find the clk50 port as shown in 18.
   - The corresponding SDC command is also shown. Press Insert and this command will be added to the SDC editor.
   - Save this file in the constraints folder with the name: lab1_gx.sdc.
   - Add this file to the Quartus project: "Project -> Add current file to project".
   - Close the SDC editor window and recompile the Quartus project.
   - Verify that the warning "Timing requirements not met" is no longer present and that the clock has been constraint to 50 MHz in the compliation report.

4. Program the FPGA.

5. Press KEY0 to start the counter. What happens? Are you able increment the value of the 7-segment display in steps of 1? If not, can you explain why?

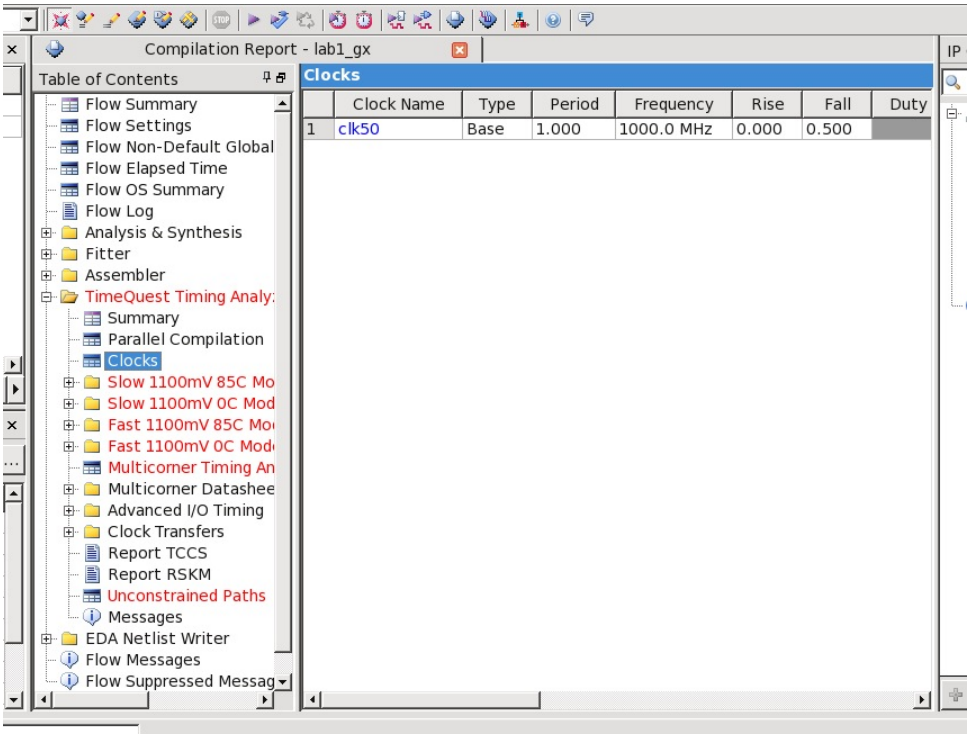

Figure 16: Compilation report showing that a clock frequency of 1000 MHz have been considered for the design.

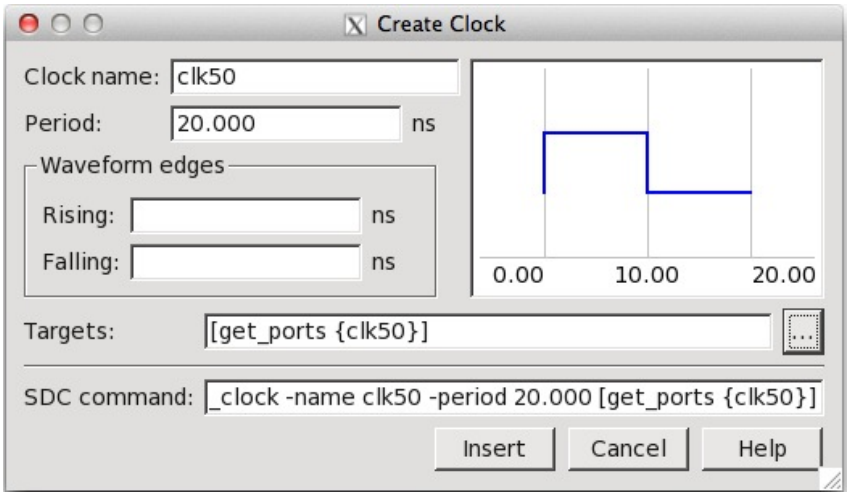

Figure 17: Constrain the clock to 50 MHz by setting a clock period of 20 ns.

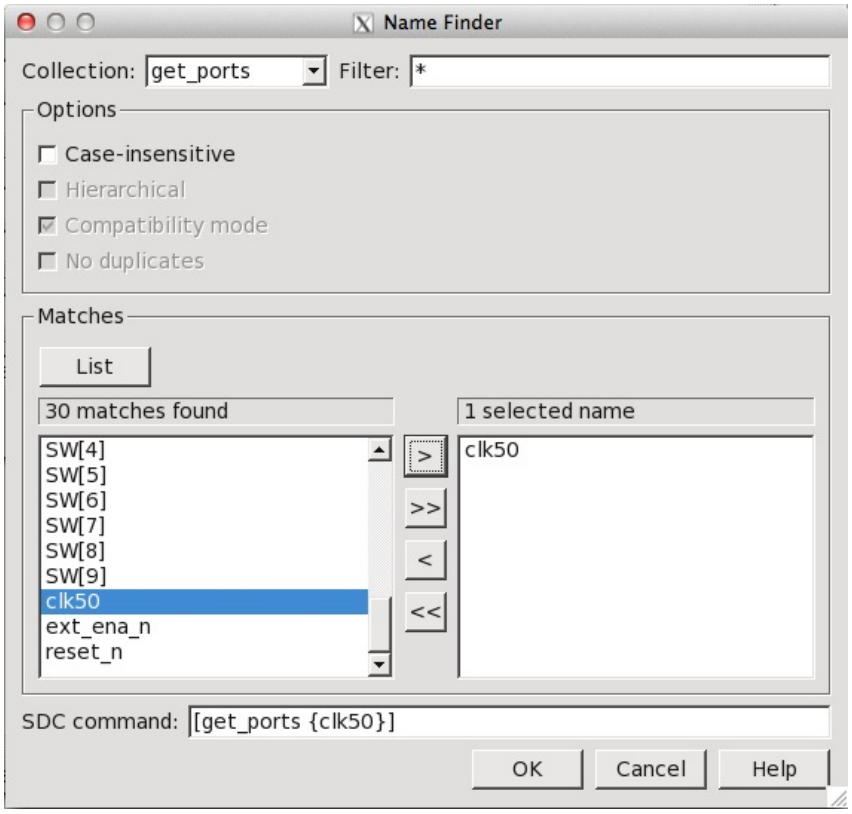

Figure 18: Find the target port to which the constraint will apply.

2.2 Edge detector

Implement the appropriate logic which is needed to control the incremental stepping of the counter and thereby the value on the 7-segment display. The design must be verified by simulation before it is downloaded to the FPGA. Provide screenshots of you simulations which clearly demonstrates the functionallity of your solution.

References

[1] DE1-SoC webpage. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=836.

[2] DE1-SoC User Manual(rev.C/rev.D Board). http://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=836&FID=ae336c1d5103cac046279ed1568a8bc3.

[3] FYS4220 Lecture 2. http://www.uio.no/studier/emner/matnat/fys/FYS4220/h15/lectures.